Deciding Substitutability of Services with Operating Guidelines

Christian Stahl*, Peter Massuthe, and Jan Bretschneider

Humboldt-Universität zu Berlin, Institut für Informatik,
Unter den Linden 6, 10099 Berlin, Germany
{stahl,massuthe,bretschn}@informatik.hu-berlin.de

Abstract. Deciding whether a service S can be substituted by another service S' is an important problem in practice and one of the research challenges in service-oriented computing. In this paper, we define three substitutability notions for services. *Accordance* specifies that S' cooperates with at least the environments that S cooperates with. S and S' are *equivalent* if they cooperate with the same environments. To guarantee that S' cooperates with a fixed subset of environments that S cooperates with, the notion of *deprecation* can be used. For each substitutability notion we present a decision algorithm. To this end we apply the concept of an *operating guideline* of a service as an abstract representation of all environments the service cooperates with.

Key words: Open nets, Operating guidelines, Service substitutability

1 Introduction

In the paradigm of service-oriented computing (SOC) [1], a service serves as a building block for designing flexible business processes by composing multiple services. Such a (composed) service is subject to changes. There may hardly ever be a total renewal or upgrade of the overall service. Instead, individual services will be replaced by better ones, because the service was too expensive or some new functionality has been added, for instance. *Service substitutability*, that is, deciding whether a service can be substituted by another service, is one of the most notable SOC research challenges [2].

Obviously, a service S can be substituted by another service S' if no environment can distinguish them, that is, they are equivalent. In practice, however, more flexible notions than equivalence are relevant as well. In general, substituting S by S' either should *gain* or *preserve* properties of the overall service.

In order to guarantee that substituting S by S' indeed gains and/or preserves specific properties, support of formal methods is needed. To this end we need to characterize different properties of substitutability, resulting in different substitutability notions. In the next step, we have to develop algorithms to decide substitutability for each notion.

* Funded by the DFG project "Substitutability of Services" (RE 834/16-1).

In this paper, we restrict ourselves to the service protocol, that is, to the *behavior* of a service, and abstract from other important aspects like quality of service and semantics. As our formal model we use *open nets*, a special class of Petri nets. An open net has an interface for communication with other open nets via asynchronous message passing. To meet different application scenarios that are relevant in practice we introduce three *substitutability notions*: accordance (S' cooperates with at least every environment S cooperates with), equivalence (S and S' cooperate with the same environments), and deprecation (S' cooperates with at least a fixed subset of environments S cooperates with). Furthermore, a property-preserving substitutability notion is derived which is more fine-grained than deprecation. For each such notion we present a decision algorithm based on the concept of an *operating guideline* as an abstract representation of all environments a given service can cooperate with. Operating guidelines have been suggested to support service discovery so far. In this paper, we show that operating guidelines are well-suited for deciding substitutability of services, too. To this end we use known results, extend some notions, and also provide new results on operating guidelines.

The remainder of this paper is structured as follows. Sections 2 and 3 present the preliminaries. There, we recall our service models, open nets and service automata, as well as operating guidelines. Then, in Sect. 4 we introduce the notion of accordance. Deprecation is explained in Sect. 5. From accordance and deprecation we derive in Sect. 6 two further substitutability notions. Related work is discussed in Sect. 7 and finally, conclusions are drawn in Sect. 8.

2 Service Models

In this section, we introduce *open nets*, a special class of Petri nets, as a formal model for services and *service automata* as a technique to analyze the interaction behavior of open nets. We will show that an open net can easily be translated into a service automaton and vice versa, so we can consider our analysis questions on both models alike.

2.1 Open Nets

We assume the usual definition of a (place/transition) *Petri net* $N = [P, T, F]$ (see [3], for instance) and use the standard notation to denote the *preset* and *postset* of a place or a transition: ${}^\bullet x = \{y \mid (y, x) \in F\}$ and $x^\bullet = \{y \mid (x, y) \in F\}$.

A *marking* of a Petri net N is a mapping $m : P \to \mathbb{N}$. We use a *multiset* notation to denote markings and write $m = [p_1, p_1, p_2]$ for a marking m with $m(p_1) = 2$, $m(p_2) = 1$, and $m(p) = 0$ for all $p \in P \setminus \{p_1, p_2\}$. If $Q \supseteq P$, a marking $m : P \to \mathbb{N}$ extends canonically to $m : Q \to \mathbb{N}$ by $m(p) = 0$ for each $p \in Q \setminus P$.

Open nets were introduced in [4] using the term "open workflow nets". Open nets are a special class of Petri nets and can be seen as a generalized version of van der Aalst's workflow nets [5], which have been proven successful for the modeling of business processes and workflows. As a substantial difference, an open net has

an *interface* that consists of a set of *input places* and a set of *output places* for asynchronous communication with an environment. This idea is based on the module concept for Petri nets which was proposed by Kindler [6]. Suitability of open nets for modeling services has been proven through an implemented translation (see [7], for instance) from the industrial service description language WS-BPEL [8] into open nets.

As a global name space, we assume a set $\mathcal{MC}$ of *message channels* given. For technical reasons, we require that the special symbols τ (representing a non-communicating step) and *final* (used to denote final states) are not in $\mathcal{MC}$.

Definition 1 (Open net).
An open net $N = [P, P_{in}, P_{out}, T, F, m_0, \Omega]$ *consists of a Petri net* $[P, T, F]$ *together with*

- *two disjoint sets* $P_{in} \subseteq (P \cap \mathcal{MC})$ *of* input places *such that* ${}^\bullet p_{in} = \emptyset$ *for all* $p_{in} \in P_{in}$ *and* $P_{out} \subseteq (P \cap \mathcal{MC})$ *of* output places *such that* $p_{out}{}^\bullet = \emptyset$ *for all* $p_{out} \in P_{out}$*,*
- *a distinguished* initial marking m_0*, and*
- *a set* Ω *of* final markings *such that no transition of* N *is enabled at any* $m \in \Omega$*.*

Let $P_{io} = P_{in} \cup P_{out}$ *denote the* interface *of* N*. We further require that neither the initial nor a final marking marks any interface place* $p \in P_{io}$*.* ⌟

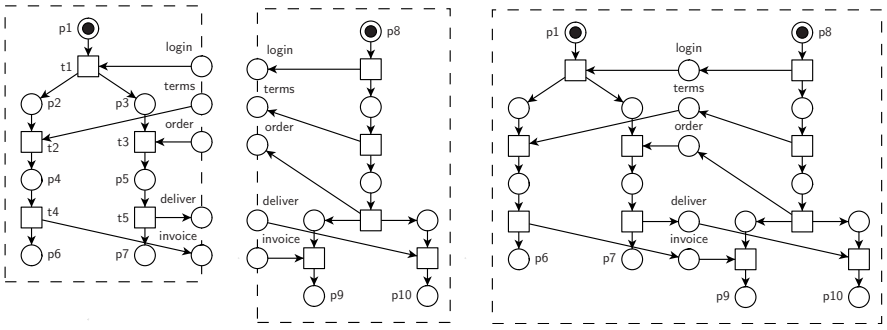
The behavior of an open net is defined using the standard Petri net semantics, that is, a transition is enabled if each place of its preset holds a token. An enabled transition t can fire in a marking m by consuming tokens from the preset places and producing tokens for the postset places, yielding a marking m'. In order to assign an intuitively consistent meaning to *final* markings, we restrict our approach to such open nets where a marking in Ω does not enable any transition.

As an example, Fig. 1(a) shows an open net model of an online shop.

Interaction of open nets is represented by their *composition*. Two open nets N_1 and N_2 are *composable* if they only share interface places and the input places of N_1 are exactly the output places of N_2 and vice versa (i.e. $P_{in1} = P_{out2}$ and $P_{in2} = P_{out1}$). For two markings m_1 of N_1 and m_2 of N_2, their *composition* $m_1 \oplus m_2$ is defined by $(m_1 \oplus m_2)(p) = m_1(p) + m_2(p)$. From now on, if two open nets N_1 and N_2 are composed, we implicitly assume they are composable.

Definition 2 (Composition of open nets).
The composition *of (composable) open nets* N_1 *and* N_2 *is the open net* $N = N_1 \oplus N_2$ *defined as follows:*

- $P = P_1 \cup P_2$*,*
- $P_{in} = P_{out} = \emptyset$*,*
- $T = T_1 \cup T_2$*,*
- $F = F_1 \cup F_2$*,*
- $m_0 = m_{0_1} \oplus m_{0_2}$*, and*
- $\Omega = \{m_1 \oplus m_2 \mid m_1 \in \Omega_1, m_2 \in \Omega_2\}$*.* ⌟



(a) Open net N_{shop} (b) Open net N_{client} (c) Composition $N_{\text{shop}} \oplus N_{\text{client}}$

Fig. 1. (a) An open net N_{shop} modeling an online shop. In the initial marking [p1], it waits for a login from a client. After the client logged in, the shop concurrently waits for an order which it then will deliver and it waits for a confirmation of the terms of payment and sends an invoice afterwards. Finally, the shop reaches the single final marking [p6, p7]. (b)-(c) An open net N_{client} modeling a client of the shop with its final marking [p9, p10] and the composition of shop and client.

A marking m of an open net N is a *deadlock* in N iff m is no final marking of N and m does not enable any transition of N. Deadlock-freedom is a fundamental correctness criterion for cooperating services. In contrast, an open net representing a service in isolation usually has deadlocks. As an example, each of the open nets in Fig. 1(a) and Fig. 1(b) on its own has at least one deadlock, whereas the open net in Fig. 1(c) is deadlock-free.

Definition 3 (Strategy).
An open net M is a (open net) strategy *for an open net N if their composition is deadlock-free. $Strat(N)$ denotes the set of all strategies for N.* ⌟

If $Strat(N) \neq \emptyset$, then N is called *controllable*, otherwise N is *uncontrollable*. Uncontrollable services are fundamentally ill-designed.

Note that according to Def. 3, the strategy notion is symmetric, that is, M is a strategy for N iff N is a strategy for M. In Sect. 3 we will show how to decide controllability of a given service N by synthesizing a strategy M, thus fixing one side of this symmetry. If N is uncontrollable, then the synthesis produces an "empty" strategy.

Obviously, the client N_{client} in Fig. 1(b) is a strategy for the shop N_{shop} in Fig. 1(a) (and vice versa). Hence, N_{shop} is controllable (and so is N_{client}).

The set $Strat(N)$ is of particular importance as it gives a semantics of an open net N in terms of N's deadlock-freely interacting environments. In Sections 4 – 6, we introduce several substitutability notions which all are based on comparing the corresponding sets of strategies.

2.2 Service Automata

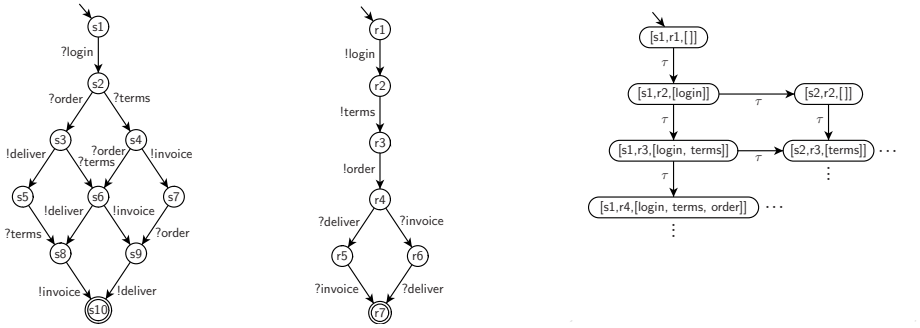
Service automata [9] form the basis of operating guidelines and are used for representing the behavior of open nets. We will firstly introduce service automata and then present a back and forth translation between open nets and service automata, that uses the set $\mathcal{MC}$ as interconnection. Service automata are closely related to the reachability graph of the inner of open nets. Thereby, the *inner* of an open net N is the open net $inner(N)$ where all interface places of N as well as all their adjacent arcs are removed.

Definition 4 (Service automaton).
A service automaton *is an automaton* $A = [Q, I_{in}, I_{out}, \delta, q_0, \Omega]$ *that consists of*

- *a set* Q *of* states*,*
- *two disjoint sets* $I_{in} \subseteq \mathcal{MC}$ *of* input channels *and* $I_{out} \subseteq \mathcal{MC}$ *of* output channels*, with* $I_{io} = I_{in} \cup I_{out}$ *is the* interface *of* A*,*
- *a nondeterministic* transition relation $\delta \subseteq Q \times (I_{io} \cup \{\tau\}) \times Q$*,*
- *a distinguished* initial state $q_0 \in Q$*, and*
- *a set* $\Omega \subseteq Q$ *of* final states*, such that* $q \in \Omega$ *and* $(q, x, q') \in \delta$ *implies* $x \in I_{in}$*.* ⌟

For a transition $(q, x, q') \in \delta$, x is called the *label* of (q, x, q'). An x-labeled transition is a *sending transition* if $x \in I_{out}$, a *receiving transition* if $x \in I_{in}$, and an *internal transition* if $x = \tau$. To emphasize the direction of an interface channel $x \in I_{io}$ in the graphical representation of a service automaton, we add an exclamation mark, $!x$, if $x \in I_{out}$, or a question mark, $?x$, if $x \in I_{in}$.

Figure 2 shows three service automata which correspond to the three open nets of Fig. 1.



(a) A_{shop} (b) A_{client} (c) Composition $A_{\text{shop}} \oplus A_{\text{client}}$

Fig. 2. Three service automata of the online shop, its client, and their composition.

In the following, we lift notions defined for open nets to service automata.

Two service automata are *composable* if they have disjoint sets of states and the input channels of one automaton are the output channels of the other automaton and vice versa. In the following, we assume all composed service automata are composable.

The *composition* $A \oplus B$ of composable service automata A and B introduces an *internal message bag* (i.e. a multiset) of currently pending messages that were sent by one automaton, but not yet received by the other one. That way, a prior x-labeled sending transition of A is represented in $A \oplus B$ by an internal (i.e. τ-labeled) transition that adds one x element to the message bag M, and a prior transition receiving an x is represented by a now internal transition removing an x from the message bag. Prior internal transitions remain as internal transitions in $A \oplus B$. This is formalized in the following definition.

Definition 5 (Composition of service automata).
For (composable) service automata A and B, their composition *is defined as the service automaton $A \oplus B = [Q, I_{in}, I_{out}, \delta, q_0, \Omega]$ defined as follows:*

- $Q = Q_A \times Q_B \times bags(\mathcal{MC})$,
- $I_{in} = I_{out} = \emptyset$,
- $\delta \subseteq Q \times \{\tau\} \times Q$,
- $q_0 = [q_{0_A}, q_{0_B}, \emptyset]$,
- $\Omega = \Omega_A \times \Omega_B \times \{\emptyset\}$,

such that the transition relation δ contains the elements

- $([q_A, q_B, M], \tau, [q'_A, q_B, M])$ *iff* $(q_A, \tau, q'_A) \in \delta_A$,
- $([q_A, q_B, M], \tau, [q_A, q'_B, M])$ *iff* $(q_B, \tau, q'_B) \in \delta_B$,
- $([q_A, q_B, M], \tau, [q'_A, q_B, M - [x]])$ *iff* $(q_A, x, q'_A) \in \delta_A$, $x \in I_{in\,A}$, $M(x) > 0$,
- $([q_A, q_B, M], \tau, [q'_A, q_B, M + [x]])$ *iff* $(q_A, x, q'_A) \in \delta_A$, $x \in I_{out\,A}$,
- $([q_A, q_B, M], \tau, [q_A, q'_B, M - [x]])$ *iff* $(q_B, x, q'_B) \in \delta_B$, $x \in I_{in\,B}$, $M(x) > 0$,
- $([q_A, q_B, M], \tau, [q_A, q'_B, M + [x]])$ *iff* $(q_B, x, q'_B) \in \delta_B$, $x \in I_{out\,B}$. ⌟

In the rest of this paper, we will only consider the connected part of the service automaton $A \oplus B$ which contains the initial state (i.e. only states which are δ-reachable from q_0).

A state q is a *deadlock* in A if $q \notin \Omega$ and at most receiving transitions leave q. Hence, a service automaton cannot leave a deadlock by its own.

Definition 6 (Strategy).
A service automaton A is a strategy *(service automaton) for a service automaton B if their composition is free of deadlocks.* ⌟

In analogy to open nets, let $Strat(A)$ denote the set of all strategies for a service automaton A. A is *controllable* iff $Strat(A) \neq \emptyset$.

2.3 Translating Open Nets into Service Automata and Back

In [10] we have shown that it is possible to transform each open net N into an open net N' where each transition is connected to at most one interface place

while preserving its set of strategies, i.e. $Strat(N) = Strat(N')$. Therefore we can, without loss of generality, assume such open nets for the transformation into service automata. Let the *inner* of such an open net N be the open net $inner(N)$ where all interface places of N as well as all their adjacent arcs are removed.

Then, the service automaton $A(N)$ of an open net N is basically the reachability graph of $inner(N)$: the states of $A(N)$ are the reachable markings of $inner(N)$ and a transition t of $inner(N)$ that was connected to an interface place p in N becomes a p-labeled transition of $A(N)$. The set $\mathcal{MC}$ is used as a common name space between the net and its corresponding service automaton, as both the interface places of N and the interface of $A(N)$ are subsets of $\mathcal{MC}$. If t was not connected to any interface place in N, then it becomes a τ-labeled (i.e. internal) transition of $A(N)$. It is easy to see that the service automata of Fig. 2 can be derived from the open nets of Fig. 1 using this transformation.

In the next section, we provide a method to synthesize a strategy service automaton B for a given (controllable) service automaton A. The value of this transformation is that, if $A = A(N)$, then each open net M with $A(M) = B$ is a strategy for N.

Additionally, it is easily possible to transform a service automaton A into an open net $N(A)$. For instance, constructing a state machine by replacing each x-labeled transition of A by a Petri net transition producing to/consuming from the interface place x. This way, we can even *construct* a strategy open net $M = N(B)$ from a strategy service automaton B.

3 Operating Guidelines

Operating guidelines were first introduced in [9] and generalized in [10]. Basically, an *operating guideline* OG_A of a service automaton A is a special service automaton B where each state q of B is annotated with a Boolean formula $\phi(q)$. Such a *Boolean annotated service automaton* (*BSA*) B^ϕ can be used to characterize a set of service automata. Therefore, we define a matching relation between a service automaton B' and a Boolean annotated service automaton B^ϕ. B^ϕ characterizes B' iff B' *matches* with B^ϕ. An operating guideline OG_A of a service automaton A is a special *BSA* where B' matches with OG_A iff B' is a strategy for A.

A *literal* of our Boolean formulae is a channel in $\mathcal{MC}$ or one of the special literals τ or *final* (representing an internal transition or a final state, respectively). Let, for the rest of this paper, $\mathcal{MC}^+$ denote the set $\mathcal{MC} \cup \{final, \tau\}$. As Boolean connectors, we only use $\vee$ (Boolean *or*) and $\wedge$ (Boolean *and*). Let $\mathcal{BF}$ be the set of all such Boolean formulae over $\mathcal{MC}^+$. As usual, we fix the truth values *true* and *false*. A (Boolean) *assignment* is a mapping $\beta : \mathcal{MC}^+ \to \{true, false\}$ assigning to each literal a truth value. Furthermore, an assignment β *satisfies* a Boolean formula $\phi \in \mathcal{BF}$ ($\beta \models \phi$) if ϕ evaluates to *true* using standard propositional logic semantics.

Definition 7 (Boolean annotated service automaton, *BSA*).
A Boolean annotated service automaton *(BSA)* $B^\phi = [Q, I_{in}, I_{out}, \delta, q_0, \Omega, \phi]$ *consists of*

- *a* deterministic *service automaton* $B = [Q, I_{in}, I_{out}, \delta, q_0, \Omega]$ *and*
- *a Boolean* annotation *function* $\phi : Q \to \mathcal{BF}$.

Thereby, a service automaton is deterministic if it has no internal transitions and each state has at most one x-labeled outgoing transition. ⌟

The restriction of *BSA*s to deterministic structures eases the decision procedures of the upcoming sections while providing all sufficient information needed for operating guidelines later on.

For matching a service automaton A with a *BSA* B^ϕ, the present outgoing transitions of a state q of A constitute an assignment for $\phi(q)$:

Definition 8 (Assignment).
An assignment *of a service automaton A assigns to each state q of A a Boolean assignment* $\beta_A(q) : \mathcal{MC}^+ \to \{\text{true}, \text{false}\}$ *defined as follows:*

$$\beta_A(q)(x) = \begin{cases} \text{true}, & \text{if } x \neq \textit{final and there is a state } q' \textit{ with } (q, x, q') \in \delta_A, \\ \text{true}, & \text{if } x = \textit{final and } q \in \Omega_A, \\ \text{false}, & \textit{otherwise.} \end{cases}$$ ⌟

A *BSA* is used to characterize a *set* of service automata. Let therefore be the matching of a service automaton with a *BSA* defined as follows:

Definition 9 (Matching).
Let A be a service automaton, B^ϕ be a BSA, and define $\varrho \subseteq Q_A \times Q_B$ inductively as follows: Let $(q_{0A}, q_{0B}) \in \varrho$. If $(q_A, q_B) \in \varrho$, $x \in \mathcal{MC}$, $(q_A, x, q'_A) \in \delta_A$, and $(q_B, x, q'_B) \in \delta_B$, then $(q'_A, q'_B) \in \varrho$. If $(q_A, q_B) \in \varrho$ and $(q_A, \tau, q'_A) \in \delta_A$, then $(q'_A, q_B) \in \varrho$.

Then, A matches *with B^ϕ if*

- *ϱ is a weak simulation relation and*
- *for each $(q_A, q_B) \in \varrho$: $\beta_A(q_A) \models \phi(q_B)$.*

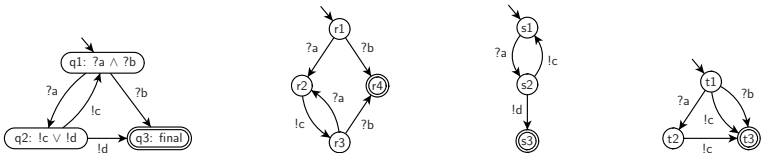
Let $Match(B^\phi)$ denote the set of all service automata that match with B^ϕ. ⌟

The weak simulation relation ϱ together with possible τ literals in ϕ allow the deterministic B^ϕ for characterizing deterministic as well as nondeterministic service automata. Figure 3(a) shows an example *BSA*. Figures 3(b) – 3(d) demonstrate the matching.

An operating guideline of a service automaton now is a special *BSA*:

Definition 10 (Operating guideline, *OG*).
A Boolean annotated service automaton $OG_A = B^\phi$ is an operating guideline *(OG) of a service automaton A iff $Match(OG_A) = Strat(A)$.* ⌟

For uncontrollable service automata A (i.e. $Strat(A) = \emptyset$) we fix an *OG* that consists of a single state that is annotated with *false*, assuring that *no* service automaton matches with this *OG*.



(a) A^{ϕ} (b) B (c) C (d) D

Fig. 3. (a) A *BSA* A^{ϕ}. The annotation $\phi(q)$ is depicted inside the state q. (b)–(d) Three service automata B, C, and D. B matches with A^{ϕ}: for instance, the assignment $\beta_B(\mathsf{r}_1)$ assigns *true* to the literals ?a and ?b (because both transitions leave the state r_1), satisfying the annotation ?a ∧ ?b. However, C and D do not match with A^{ϕ}: state s_1 does not satisfy the annotation of state q_1; and the !c-labeled transition leaving state t_1 causes A not simulating D.

Figure 4 depicts an operating guideline of our online shop example of Fig. 2(a). It is easy to see that the client of Fig. 2(b) matches with this OG.

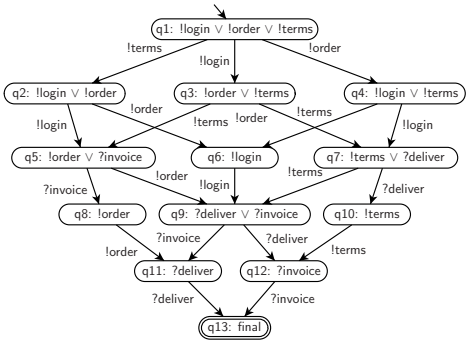


Fig. 4. An operating guideline of the online shop of Fig. 2(a). To characterize also nondeterministic strategies, each Boolean annotation $\phi(q)$ is implicitly extended to $\phi(q) \vee \tau$ and thus evaluated to *true* if the matched service automaton has an outgoing τ-transition in the corresponding state (cp. Def. **??**).

In [10] we have presented an algorithm to compute an operating guideline of a service where the inner of the service (cp. Sect. 2.3) has finitely many reachable states. For services without this restriction, we were able to show that controllability is undecidable [11]. The algorithm is constructive, i.e. it computes a special strategy. Therefore it starts with an overapproximation of compatible behavior of any strategy removing deadlock-causing states iteratively. If the service is uncontrollable, the algorithm eventually removes all states. The algorithm is implemented in our tool Fiona[1] [7].

4 Accordance

In this section, we define our first substitutability notion, *accordance*. A service A' accords with a service A if A' cooperates with at least the environments that

[1] Fiona is available at http://www.informatik.hu-berlin.de/top/tools4bpel.

A cooperates with. That is, if the composition of A and an environment B is deadlock-free, then deadlock-freedom is preserved if A is substituted by A'.

4.1 A Notion of Accordance

Given a service automaton A, it might be necessary to change or add some functionality of A by substituting it by a new version A'. With accordance, we demand that this substitution must not affect any customer of A: every current customer of A has to be supported by A' as well. Because we assume that A does not know each customer that uses A, A' must support each *potential* customer of A, i.e. all elements in $\mathit{Strat}(A)$. An application for accordance is the upgrade of a web shop which should not affect any customer. This motivates the following notion of accordance between service automata A and A'. To this end A and A' must be *interface equivalent* (i.e. $I_{in\,A} = I_{in\,A'}$ and $I_{out\,A} = I_{out\,A'}$).

Definition 11 (Accordance).
Let A and A' be interface equivalent service automata. A' substitutes A under accordance *(short: A' accords with A) iff* $\mathit{Strat}(A) \subseteq \mathit{Strat}(A')$. ⌟

Accordance guarantees that every strategy for A is a strategy for A' as well. In other words, if A' accords with A, then every customer of A is also a customer of A'. In addition, accordance allows for new customers of A'. Thus, accordance seems to be the right notion to achieve the goal mentioned above.

The notion of accordance has been first introduced in [12, 13]. However, the decision procedure for accordance was limited to acyclic finite state services there. In this paper, we extend this procedure to cyclic finite state services.

4.2 Deciding Accordance

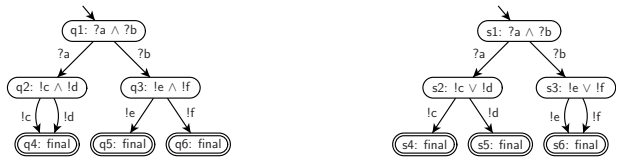
In order to decide accordance of A and A', we need to compare $\mathit{Strat}(A)$ and $\mathit{Strat}(A')$. The problem is that the set *Strat* may correspond to a large (in fact infinite) set of service automata. With the operating guidelines of A and A' we have, however, a finite representation of $\mathit{Strat}(A)$ and $\mathit{Strat}(A')$. In the following, we show how accordance can be decided by using operating guidelines. To this end we define a refinement relation $\sqsubseteq$ for operating guidelines. Informally, $OG_A \sqsubseteq OG_{A'}$, that is, $OG_{A'}$ refines OG_A iff there is a simulation relation between the states of OG_A and $OG_{A'}$ such that the annotations in OG_A imply the annotations in $OG_{A'}$.

Definition 12 ($\sqsubseteq$-relation of OGs).
Let A and A' be interface equivalent service automata and let $OG_A = [Q, I_{in}, I_{out}, \delta, q_0, \Omega, \phi]$ and $OG_{A'} = [Q', I'_{in}, I'_{out}, \delta', q'_0, \Omega', \phi']$ be the corresponding operating guidelines. Then, $OG_A \sqsubseteq OG_{A'}$ iff there is a relation $\xi \subseteq Q \times Q'$ such that

1. $(q_0, q'_0) \in \xi$;
2. *if $(q, q') \in \xi$ and $(q, x, q_1) \in \delta$, then there is a q'_1 such that $(q', x, q'_1) \in \delta'$ and $(q_1, q'_1) \in \xi$; and*

3. for all $(q, q') \in \xi$, the formula $\phi(q) \Rightarrow \phi'(q')$, is a tautology. ⌟

The relation $\sqsubseteq$ is a preorder, that is, it is reflexive and transitive. By help of the next theorem we show that $OG_{A'}$ refines OG_A iff A' accords with A and thus it can be used to decide accordance of A and A'. An example is depicted in Fig. 5.



(a) OG_{small} (b) OG_{big}

Fig. 5. Two operating guidelines with $OG_{\text{small}} \sqsubseteq OG_{\text{big}}$. For instance, $(\mathsf{q2}, \mathsf{s2}) \in \xi$ with $\phi(\mathsf{q2}) \Rightarrow \phi(\mathsf{s2})$, and $(\mathsf{q4}, \mathsf{s4}) \in \xi$ and $(\mathsf{q4}, \mathsf{s5}) \in \xi$ with $\phi(\mathsf{q4}) \Rightarrow \phi(\mathsf{s4})$ and $\phi(\mathsf{q4}) \Rightarrow \phi(\mathsf{s5})$.

Theorem 1 (Checking accordance).
Let A and A' be two service automata and let OG_A and $OG_{A'}$ be the corresponding operating guidelines. Then, $OG_A \sqsubseteq OG_{A'}$ iff $Strat(A) \subseteq Strat(A')$. ⌟

For the proof of this theorem, we rely on a fact about operating guidelines as constructed in [10]. As we cannot repeat the whole approach of [10], we only include the following proposition and then sketch the proof of Thm. 1.

Proposition 1 ([10]).
For every operating guideline $OG_A = [Q, I_{in}, I_{out}, \delta, q_0, \Omega, \phi]$ (of some controllable service automaton A) and all $q \in Q$, the formula $\phi(q)$

1. *uses only literals x where there is some $q' \in Q$ with $(q, x, q') \in \delta$, and*
2. *is satisfied for the assignment assigning* true *to all literals in $\phi(q)$.* ⌟

Proof (of Thm. 1 (Sketch)).
Let $OG_A = [Q, I_{in}, I_{out}, \delta, q_0, \Omega, \phi]$ and $OG_{A'} = [Q', I'_{in}, I'_{out}, \delta', q'_0, \Omega', \phi']$ be the operating guidelines of service automata A and A', respectively.

Implication. *Let $OG_A \sqsubseteq OG_{A'}$ and let B be an arbitrary strategy service automaton for A. We show that B is a strategy for A', too.*

By Def. 9, there is a simulation relation $\varrho \subseteq Q_B \times Q$ between (the states of) B and OG_A and, by Def. 12, there is a relation $\xi \subseteq Q \times Q'$ between OG_A and OG_B. Let $\varrho' \subseteq Q_B \times Q'$ be a relation between B and $OG_{A'}$ defined as follows: $(q_B, q') \in \varrho'$ iff there is a state q of OG_A such that $(q_B, q) \in \varrho$ and $(q, q') \in \xi$.

Obviously, ϱ' is a simulation relation between B and $OG_{A'}$. Let $(q_B, q') \in \xi$. Because B matches with OG_A, it holds for all states q_B with $(q_B, q) \in \varrho$ that $\phi(q)$ evaluates to true *for the assignment described in Def. 9. Because of $OG_A \sqsubseteq OG_{A'}$, every such assignment satisfies also $\phi(q')$ for $(q, q') \in \xi$. Hence, q_B satisfies $\phi(q')$ and, consequently, B is a strategy for A', too.*

Replication. *Let $Strat(A) \subseteq Strat(A')$. Consider the underlying service automaton $B = [Q, I_{in}, I_{out}, \delta, q_0, \Omega]$ of OG_A. By construction, the transition systems of B and OG_A are equivalent and hence there is a weak simulation relation between the states of B and OG_A. Furthermore, as there is a transition in B for each $(q, x, q') \in \delta$ in OG_A, we can derive from Prop. 1 that all annotations evaluate to* true *when B is evaluated according to Def. 9. Consequently, B matches with OG_A and hence B is a strategy for A and thus, by assumption, a strategy for A'.*

Being a strategy for A', there is a relation ϱ' between the states of B and $OG_{A'}$. Let $q \in Q$. Define $\xi \subseteq Q \times Q'$ such that $\xi(q)$ is the set of states in Q' that is equivalent to the union of $\varrho'(q_B)$, for all $q_B \in \varrho(q)$.

By the structural similarity of Def. 9 and Def. 12, it is easy to see that ξ satisfies the first two items required in Def. 12. For verifying the third item, let $q \in Q$ and let β be an arbitrary assignment to literals occurring in $\phi(q)$ where $\phi(q)$ is true. *Remove from B all those transitions (q_1, x, q_2) where $\beta(x)$ is* false. *By Def. 9, the resulting service automata is still a strategy for A and thus a strategy for A', too. Using Def. 9 again, we can see that $\phi'(q')$ is* true *as well for all $q' \in \xi(q)$. Thus, $\phi(q) \Rightarrow \phi'(q')$, $(q, q') \in \xi$, is a tautology.*

The value of this theorem is that accordance can be checked independently of the environments that A cooperates with and only A and A' have to be known to decide accordance. In order to design a service automaton A' which accords with A, a designer can either try and check the resulting service or he derives A' from A by applying accordance-preserving transformation rules [12].

For an implementation of the criteria in Thm. 1, finding the relation ξ is the crucial task. As both OG_A and $OG_{A'}$ are deterministic, this task actually amounts to a depth-first search through $OG_{A'}$ which is mimicked in OG_A. The time and space required for finding ξ is thus linear in the number of states and edges of $OG_{A'}$. This size, in turn, is equal to the number of states and edges of a particular strategy for A [14]. The accordance check based on Thm. 1 has been implemented in our tool Fiona.

5 Deprecation

In this section, we introduce another substitutability notion, *deprecation*. Deprecation is – as accordance – used to compare the sets of environments of two service automata A and A'. The goal of deprecation is to preserve at least a *fixed subset* of the environments of A by A' (instead of *all* environments of A as in the accordance setting).

5.1 A Notion of Deprecation

Given a service automaton A, we may want to preserve at least a fixed subset $\mathcal{S} \subseteq Strat(A)$ of its strategies when substituting A by a service automaton A'. This means, every service automaton $S \in \mathcal{S}$ is a strategy for both A and A'.

In contrast to the notion of accordance, here we assume that A *has* knowledge of its environments. To motivate the need of such a substitutability notion, consider again an upgrade of a web shop. Applications for deprecation include: the upgraded shop only supports behavior which is used by major customers and all other customers have to adjust their services; the shop restricts itself to its core competencies and rejects all unprofitable strategies; the shop restricts its behavior to certain scenarios such as payment via VISA, for instance. These considerations lead us to the following definition of deprecation.

Definition 13 (Deprecation).
Let A and A' be interface equivalent service automata and let $\mathcal{S} = \{S_1, \ldots, S_n\} \subseteq Strat(A)$. Then, A' substitutes A under deprecation preserving $\mathcal{S}$ *(short: A' preserves $\mathcal{S}$) iff $\mathcal{S} \subseteq Strat(A')$.* ⌟

According to this definition, at least every service automaton in $\mathcal{S}$ is a strategy for A', meaning, the substitution preserves at least strategies $\mathcal{S}$. Hence, deprecation seems to be the right notion to achieve the above mentioned goal.

5.2 Deciding Deprecation

The aim of this section is to introduce a decision procedure whether substituting a service automaton A by a service automaton A' preserves a set $\mathcal{S} \subseteq Strat(A)$ of strategies. Therefore we have to check that every service automaton $S \in \mathcal{S}$ is a strategy for A'. This decision procedure becomes particularly complex if the set $\mathcal{S}$ contains many service automata and we want to check several A'. Therefore, we consider the following alternative: since the notion of a strategy is symmetric, it is equivalent to check whether A' is a strategy for all $S \in \mathcal{S}$. In other words, $A' \in \bigcap_{S \in \mathcal{S}} Strat(S)$ must hold.

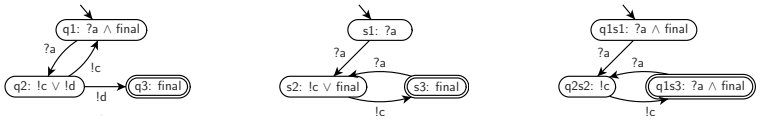
We will show that the intersection $\bigcap_{S \in \mathcal{S}} Strat(S)$ of sets of strategies can be represented by the *product of the operating guidelines* of all service automata $S \in \mathcal{S}$. We start by defining the product $OG_A \otimes OG_B$ of two operating guidelines OG_A and OG_B of service automata A and B as an operating guideline which characterizes exactly the intersection $Strat(A) \cap Strat(B)$. To this end, OG_A and OG_B must be interface equivalent, that is, their underlying automata must be interface equivalent.

Definition 14 (Product of OGs).
For two interface equivalent operating guidelines $OG_A = C_1^{\phi_1}$ and $OG_B = C_2^{\phi_2}$ with $C_1 = [Q_1, I_{in1}, I_{out1}, \delta_1, q_{01}, \Omega_1, \phi_1]$ and $C_2 = [Q_2, I_{in2}, I_{out2}, \delta_2, q_{02}, \Omega_2, \phi_2]$ their product *$OG_A \otimes OG_B = [Q, I_{in}, I_{out}, \delta, q_0, \Omega, \phi]$ is defined by*

- *$Q = \varrho$ with ϱ is the matching relation between states of C_1 and C_2,*
- *$I_{in} = I_{in1} = I_{in2}$,*
- *$I_{out} = I_{out1} = I_{out2}$,*
- *$((q_1, q_2), x, (q_1', q_2')) \in \delta$ iff $(q_1, x, q_1') \in \delta_1$ and $(q_2, x, q_2') \in \delta_2$,*
- *$q_0 = (q_{01}, q_{02})$,*
- *$\Omega = \{(q_1, q_2) \in Q \mid q_1 \in \Omega_1, q_2 \in \Omega_2\}$, and*

– $\phi((q_1, q_2)) = \phi_1(q_1) \wedge \phi_2(q_2)$, *for all* $(q_1, q_2) \in Q$. ⌟

In a way, the product of operating guidelines is defined analogously to the product of finite automata. Figure 6 shows two operating guidelines and their product (with $\varrho = \{(\mathsf{q1}, \mathsf{s1}), (\mathsf{q2}, \mathsf{s2}), (\mathsf{q1}, \mathsf{s3})\}$).



(a) OG_1 (b) OG_2 (c) $OG_1 \otimes OG_2$

Fig. 6. Two operating guidelines and their product.

Next, we prove that the product of two operating guidelines characterizes indeed the intersection of the strategies represented by these operating guidelines. For the proof we make use of the following lemma.

Lemma 1.
Let C be a service automaton, OG_A and OG_B be operating guidelines, and let $OG_\otimes = OG_A \otimes OG_B$ be their product. Let ϱ_{CA}, ϱ_{CB} and $\varrho_{C\otimes}$ denote the matching relations between C and the respective annotated automaton.

Then, for all $q_C \in Q_C$, $q_A \in Q_A$, $q_B \in Q_B$ holds: $(q_C, (q_A, q_B)) \in \varrho_{C\otimes}$ iff $(q_C, q_A) \in \varrho_{CA}$ and $(q_C, q_B) \in \varrho_{CB}$. ⌟

Proof.
Let $q_C \in Q_C$, $q_A \in Q_A$, $q_B \in Q_B$. It holds:

$(q_C, (q_A, q_B)) \in \varrho_\otimes$
iff there is a sequence σ of message channels such that:
q_C *is reached from q_{0C} by following δ_C along σ and (q_A, q_B) is reached from (q_{0A}, q_{0B}) by following $\delta_\otimes$ along σ (by Def. 9),*
iff q_A *is reached from q_{0A} by following δ_A along σ and*
q_B *is reached from q_{0B} by following δ_B along σ (by Def. 14),*
iff $(q_C, q_A) \in \varrho_A$ *and* $(q_C, q_B) \in \varrho_B$ *(by Def. 9).*

Thus, the lemma holds.

Theorem 2 (Product OG characterizes intersection).
Let $OG_\otimes = OG_A \otimes OG_B$ be the product of operating guidelines OG_A and OG_B. Then, $Match(OG_\otimes) = Match(OG_A) \cap Match(OG_B)$. ⌟

Proof.
Let $OG_A = [Q_A, I_{in\,A}, I_{out\,A}, \delta_A, q_{0A}, \Omega_A, \phi_A]$, $OG_B = [Q_B, I_{in\,B}, I_{out\,B}, \delta_B, q_{0B}, \Omega_B, \phi_B]$ and $OG_\otimes = OG_A \otimes OG_B = [Q, I_{in}, I_{out}, \delta, q_0, \Omega, \phi]$.

Implication. *Let $C \in Match(OG_\otimes)$. We will show that $C \in Match(OG_A)$ and $C \in Match(OG_B)$, too. Let $(q_C, q_A) \in \varrho_{CA}$ and $(q_C, q_B) \in \varrho_{CB}$. According to Lemma 1 we have $(q_C, (q_A, q_B)) \in \varrho_{C\otimes}$. Let $x \in \mathcal{MC} \cup \{\tau\}$ and let there be an x-transition leaving q_C. From $C \in Match(OG_\otimes)$ and from Def. 9 (i.e. the weak*

simulation relation) we can conclude, there is an x-transition leaving (q_A, q_B), too. By the construction of δ in Def. 14, there is an x-transition leaving q_A and leaving q_B.

Furthermore, we conclude from $C \in Match(OG_\otimes)$ and Def. 9 that the assignment $\beta_C(q_C)$ satisfies $\phi((q_A, q_B))$. Hence, by the construction of ϕ in Def. 14, $\beta_C(q_C)$ also satisfies $\phi_A(q_A)$ and $\phi_B(q_B)$. Consequently, C matches with OG_A and OG_B and therefore $C \in Match(OG_A) \cap Match(OG_B)$.

Replication. *Let $C \in Match(OG_A) \cap Match(OG_B)$. We will show that $C \in Match(OG_\otimes)$, too. Let $(q_C, q) \in \varrho_{C\otimes}$ with $q = (q_A, q_B)$. According to Lemma 1 we have $(q_C, q_A) \in \varrho_{CA}$ and $(q_C, q_B) \in \varrho_{CB}$. Let $x \in \mathcal{MC} \cup \{\tau\}$ and let there be an x-transition in q_C. From $C \in Match(OG_A) \cap Match(OG_B)$ and from Def. 9 (i.e. the weak simulation relation), there is an x-transition in q_A and in q_B. By the construction of δ in we can conclude Def. 14, there is an x-transition in q.*

Furthermore, we conclude from $C \in Match(OG_A) \cap Match(OG_B)$ and Def. 9 that the assignment $\beta_C(q_C)$ satisfies $\phi_A(q_A)$ and $\phi_B(q_B)$. Hence, by the construction of ϕ in Def. 14, $\beta_C(q_C)$ also satisfies $\phi((q_A, q_B))$. Consequently, C matches with $OG_\otimes$ and therefore $C \in Match(OG_\otimes)$.

The product $\otimes$ of operating guidelines is commutative and associative, that is, for operating guidelines OG_A, OG_B, OG_C holds $OG_A \otimes OG_B = OG_B \otimes OG_A$ and $(OG_A \otimes OG_B) \otimes OG_C = OG_A \otimes (OG_B \otimes OG_C)$. Thus, we conclude that $OG_\otimes$ represents exactly the intersection of all sets of strategies for services automata in $\mathcal{S}$:

Corollary 1.
Let $\mathcal{S} = \{S_1, \ldots, S_n\}$ be a set of interface equivalent service automata and let OG_{S_i} be the operating guideline of S_i, for all $1 \leq i \leq n$. Let $OG_\otimes$ denote the product of all OG_{S_i}. Then, $Match(OG_\otimes) = \bigcap_{S \in \mathcal{S}} Strat(S)$. ⌟

With the help of the above corollary we can prove a theorem which shows that substituting A by A' preserves $\mathcal{S}$ iff A' is a strategy represented by the product operating guideline $OG_\otimes$.

Theorem 3 (Deprecation check with product OGs).
Let A and A' be service automata and let $\mathcal{S} = \{S_1, \ldots, S_n\} \subseteq Strat(A)$. Let OG_{S_i}, $1 \leq i \leq n$, be the operating guideline of S_i and let $OG_\otimes$ denote the product of all OG_{S_i}. Then, A' preserves $\mathcal{S}$ iff $A' \in Match(OG_\otimes)$. ⌟

Proof.
We will show that $Match(OG_\otimes)$ characterizes all service automata A' that can substitute A while preserving $\mathcal{S}$. We have:

$$\begin{aligned} Match(OG_\otimes) &= \bigcap_{S \in \mathcal{S}} Strat(S) \text{ (Cor. 1)} \\ &= \{A' \mid \text{for all } S \in \mathcal{S} : A' \in Strat(S)\} \\ &= \{A' \mid \text{for all } S \in \mathcal{S} : S \in Strat(A')\} \text{ (strategy is symmetric)} \\ &= \{A' \mid A' \text{ preserves } \mathcal{S}\} \text{ (Def. 13)} \end{aligned}$$

Consequently, the theorem holds.

In order to decide whether substituting A by A' preserves $\mathcal{S} \subseteq \mathit{Strat}(A)$, we have to construct the operating guideline for each $S \in \mathcal{S}$ and then calculate the product of these operating guidelines. Time and space complexity for calculating the product of two operating guidelines is proportional to the product of their states. Therefore, this complexity effort only pays off if we check several A'. The deprecation check based on Thm. 3 has been implemented in our tool FIONA.

Intuitively, the fewer strategies shall be preserved by the substitution (i.e. the smaller $\mathcal{S}$ is), the more service automata A' exist that may substitute A (i.e. the bigger is $\mathit{Match}(OG_{\otimes})$). Because accordance requires all strategies for A to be preserved by A', but deprecation requires only a subset of A's strategies to be preserved by A', there are less services A' that accord with A, than services A' that satisfy deprecation. Note that for $\mathcal{S} = \mathit{Strat}(A)$ deprecation coincides with accordance.

As an advantage, the notion of deprecation provides with $OG_{\otimes}$ an abstract representation of all substitutable service automata A'. In case of accordance, in contrast, A' has either to be guessed or derived by applying accordance-preserving transformation rules.

6 Derived Substitutability Notions

In this section, we introduce two more substitutability notions. Both notions can be derived from the notions of accordance and deprecation.

6.1 Equivalence

The first substitutability notion we derive is a notion of *equivalence* for service automata. This can be achieved easily by restricting the notion of accordance. Two service automata are equivalent iff they have the same set of strategies.

Definition 15 (Equivalence).
Let A and A' be interface equivalent service automata. Then, A' equivalently substitutes *A (short: A' and A are equivalent) iff $\mathit{Strat}(A) = \mathit{Strat}(A')$.* ⌟

Obviously, in order to check equivalence of two service automata, we can check equivalence of their respective operating guidelines. Since equivalence means accordance in both directions, we apply Thm. 1 in both directions.

Corollary 2 (Checking equivalence with OGs).
Two operating guidelines OG_A and OG_B are equivalent, denoted $OG_A \equiv OG_B$, iff $OG_A \sqsubseteq OG_B$ and $OG_A \sqsupseteq OG_B$. ⌟

6.2 Constraints

For many substitutability scenarios the three notions of substitutability we have introduced so far are well-suited. However, there are other scenarios in practice

that require less restrictive notions. Accordance demands to preserve all strategies for a given service, even those which are practically infeasible: consider that a service A has to interact with *two* other services, B and C. Assume that A sends a request to either service B or C and concurrently expects an acknowledgement from the respective service. There is a strategy S for A such that S receives the request which A has sent to B and acknowledges on behalf of C. This is, in fact, a valid strategy, but practically impossible if B and C do not communicate with each other. This problem arises in the decentralized setting [15]. Such strategies need not to be preserved when substituting A by A'.

As another example, if we want to restrict the set of strategies to profitable strategies or to enforce or exclude certain scenarios (e.g. payment via VISA), then deprecation is too inflexible, because we would have to identify all infeasible strategies.

These examples motivate the introduction of a notion of a *constraint*. Such a constraint can been seen as a behavioral pattern or communication scenario. We will show how to restrict a set of strategies to those strategies that *enforce* or *exclude* certain behavioral patterns. In [16] such constraints have been introduced to characterize all strategies for a service that conform to a constraint. This approach is used to filter *service registries* for services that fit respective strategies and for validating services by checking whether there exist strategies that access certain features. In contrast to [16], we are interested in services that preserve all strategies that conform to a constraint.

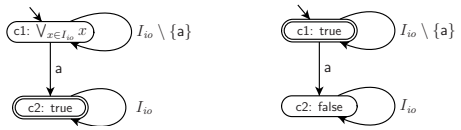
In the following, we define the notion of a constraint *BSA* C^ψ. Intuitively, C^ψ is a *BSA* that constrains send and receive actions of an operating guideline OG_A. Here, to constrain means to enforce or to exclude the respective actions of OG_A.

Definition 16 (Constraint *BSA*).
Let A and C be two interface equivalent service automata. Let OG_A be an operating guideline of A and let ψ be an annotation to C. Then, C^ψ is a constraint *BSA for OG_A.* ⌟

Intuitively, OG_A represents the set of strategies for A and the constraint *BSA* C^ψ describes the behavior we want to allow or disallow in the restricted subset of strategies. Therefore, their product characterizes all strategies for A that conform to C^ψ. Figure 7 depicts generic constraint automata for enforcing or excluding a communication action a.

Given a product $OG_A \otimes C^\psi$, each service automaton A' where $OG_{A'}$ characterizes exactly these strategies is a well-suited candidate for substituting A. This yields a more fine-grained notion of substitutability under deprecation which is covered by the following corollary.

Corollary 3 (Constraint-conforming substitution).
Let A, A' be service automata and OG_A, $OG_{A'}$ be the corresponding operating guidelines. Let C^ψ be a constraint BSA for OG_A. Then, the substitution of A by A' conforms to C^ψ *iff* $Match(OG_{A'}) = Match(OG_A \otimes C^\psi)$. ⌟



(a) *enforce*(a) (b) *exclude*(a)

Fig. 7. Generic constraint automata to enforce or exclude a communication action a.

In order to apply the results presented in this section, a designer has either to construct or to guess a service automaton A'. The correctness of A' can then be checked by applying Cor. 3. The notion of constraints and the substitutability check based on Cor. 3 has been implemented in our tool Fiona.

7 Related Work

Various substitutability notions can be found in literature. However, most of them lack of an asynchronous communication model as it is necessary in the context of SOC or efficient decision algorithms; or they are restricted to an equivalence notion.

Vogler presents in [17] a livelock and deadlock preserving equivalence between Petri nets with interfaces. However, there is no direct implication in either direction between the equivalence of Vogler and accordance.

For workflow nets (WFNs) [5] the notion of *inheritance* [18, 19] is used two relate two WFNs that can be substituted. Inheritance bases on branching bisimulation. As a difference, the inheritance approach assumes a synchronous communication model (i.e. transition fusion). Furthermore, in [12], our notion of accordance has been proven to be more liberal than the notion of projection inheritance, that is, projection inheritance implies accordance.

Bonchi et al. [20] also model the behavior of services with Petri nets. They propose *saturated bisimulation* as equivalence notion which is, however, too restrictive to allow reordering of messages (in contrast to our equivalence notion).

In [21–24] automata models are used to decide substitutability. All these approaches use only synchronous communication whereas we consider asynchronous message passing. Benatallah et al. [23] present four notions of substitutability. In this paper, we cover all of them. Equivalence and subsumption mean in our notion equivalence and accordance. In the third notion, service S can be substituted by S' assuming the environment E is known. In this setting, we would check whether S is a strategy for E. Finally, in the fourth notion, S is substituted by S' w.r.t. an interaction role R, that is, the intersection of S and R has to behave as S'. In our notion we would check if $OG_R \otimes OG_S \sqsubseteq OG_{S'}$.

Refinement relations similar to our notion of accordance are also used in the setting of service contracts. Several refinement relations, called *conformance*, have been proposed in literature.

Castagna et al. [25] introduce a conformance notion for finite-state systems that formalizes like our notion of accordance the absence of deadlocks and in addition livelocks. In contrast to accordance and other conformance notions,

conformance in [25] only demands the termination of the environment but not the termination of the process itself.

In [26] Bravetti and Zavattaro propose a conformance notion that guarantees the absence of deadlocks, livelocks, and infinite runs in cyclic systems.

As the main difference to our notion of accordance, [25, 26] define their notions for synchronous communication and they do not explicitly show how asynchronous message passing can be translated into their calculi although it seems to be possible in general.

Fournet et al. [27] present stuck-free conformance, a refinement relation between two CCS processes of asynchronous message passing software components. Stuck-freedom formalizes the absence of deadlocks. To check conformance, the model checker Zing [28] is used. Stuck-free conformance requires among others that an implemented process S' simulates its original process S. Our approach, in contrast, requires a simulation relation between operating guidelines of S and S', that is, we do not compare S and S', but their strategies. It seems that our notion of accordance is more general than stuck-free conformance.

The *ComFoRT* framework [29] analyzes whether a software component S implemented in the programming language C can be substituted by another software component S'. S can be substituted by S' if the following two criteria hold: (i) Every behavior possible in S must also be a behavior of S', and (ii) the new version of the software system must satisfy previously established correctness properties. This notion coincides with our notion of accordance.

Pathak et al. [30] focus on a substitutability notion that preserves certain properties of a service S to be substituted. The properties are expressed by a μ-calculus formula ϕ. Then, a μ-calculus formula ψ is calculated such that all services S' that satisfy ψ can substitute S. Due to the expressiveness of the μ-calculus in comparison to our proposed constraints on visible actions of open nets, this approach generalizes of our property-preserving substitution, but it assumes, however, a synchronous communication model.

8 Conclusion

We have investigated the problem whether a service S can be substituted by another service S'. Based on our formal models of open nets and service automata, we have defined different substitutability notions for services: accordance, deprecation (in two variants), and equivalence. That way we can formally support various substitutability scenarios which may occur in practice.

As our substitutability notions compare the infinite sets of all deadlock-freely interacting services for S and S', the presented decision algorithms apply the concept of an operating guideline as a finite representation of these sets of services. That way we can decide accordance and equivalence for S and S'. In addition, we defined the notion of a product operating guideline to specify the intersection of the services represented by several operating guidelines. Product operating guidelines are well-suited to characterize all deadlock-freely interacting services

for a fixed set of services and can therefore be used for deciding deprecation and the more fine-grained deprecation notion of property-preserving substitutability.

We implemented all decision algorithms presented in this paper in our analysis tool Fiona. The main functionality of Fiona is to calculate an operating guideline of a service modeled as an open net. With the help of the compiler BPEL2oWFN we can translate WS-BPEL processes into our formal model open nets. Using Fiona we can decide on the Petri net model whether these WS-BPEL processes can be substituted according to one of the substitutability notions presented in this paper. That way we can apply our results to practical applications.

In ongoing research, we will work on other termination criteria than deadlock-freedom. This includes the absence of livelocks and the absence of infinite runs.

References

1. Papazoglou, M.P.: Web Services: Principles and Technology. Pearson - Prentice Hall, Essex (2007)
2. Papazoglou, M.P., Traverso, P., Dustdar, S., Leymann, F., Krämer, B.J.: 05462 Service-Oriented Computing: A Research Roadmap. In Curbera, F., Krämer, B.J., Papazoglou, M.P., eds.: Service Oriented Computing (SOC), 15.-18. November 2005. Volume 05462 of Dagstuhl Seminar Proceedings., Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany (2006)
3. Reisig, W.: Petri Nets. EATCS Monographs on Theoretical Computer Science edn. Springer-Verlag (1985)
4. Massuthe, P., Reisig, W., Schmidt, K.: An Operating Guideline Approach to the SOA. Annals of Mathematics, Computing & Teleinformatics **1**(3) (2005) 35–43
5. Aalst, W.v.d.: The application of Petri nets to workflow management. Journal of Circuits, Systems and Computers **8**(1) (1998) 21–66
6. Kindler, E.: A compositional partial order semantics for Petri net components. In: 18th International Conference on Applications and Theory of Petri Nets and Other Models of Concurrency (ICATPN 1997). Volume 1248 of Lecture Notes in Computer Science., Springer-Verlag (1997) 235–252
7. Lohmann, N., Massuthe, P., Stahl, C., Weinberg, D.: Analyzing Interacting BPEL Processes. In Dustdar, S., Fiadeiro, J.L., Sheth, A., eds.: 4th International Conference on Business Process Management (BPM 2006). Volume 4102 of Lecture Notes in Computer Science., Vienna, Austria, Springer-Verlag (2006) 17–32
8. Alves, A., et al.: Web Services Business Process Execution Language Version 2.0. Committee Specification, Organization for the Advancement of Structured Information Standards (OASIS) (2007)
9. Massuthe, P., Schmidt, K.: Operating Guidelines - An Automata-Theoretic Foundation for the Service-Oriented Architecture. In Cai, K.Y., Ohnishi, A., Lau, E.M.F., eds.: 5th International Conference on Quality Software (QSIC 2005), Melbourne, Australia, IEEE Computer Society (2005) 452–457
10. Lohmann, N., Massuthe, P., Wolf, K.: Operating Guidelines for Finite-State Services. In Kleijn, J., Yakovlev, A., eds.: 28th International Conference on Applications and Theory of Petri Nets and Other Models of Concurrency (ICATPN 2007). Volume 4546 of Lecture Notes in Computer Science., Siedlce, Poland, Springer-Verlag (2007) 321–341

11. Massuthe, P., Serebrenik, A., Sidorova, N., Wolf, K.: Can I find a partner? Preprint CS-01-08, Universität Rostock, Rostock, Germany (2008)
12. Aalst, W.M.P.v.d., Lohmann, N., Massuthe, P., Stahl, C., Wolf, K.: From Public Views to Private Views – Correctness-by-Design for Services. In Dumas, M., Heckel, R., eds.: Web Services and Formal Methods, Forth International Workshop, WS-FM 2007 Brisbane, Australia, September 28-29, 2007, Proceedings. Volume 4937 of Lecture Notes in Computer Science., Springer-Verlag (2008) 139–153
13. Aalst, W.M.P.v.d., Massuthe, P., Stahl, C., Wolf, K.: Multiparty Contracts: Agreeing and Implementing Interorganizational Processes. Informatik-Berichte 213, Humboldt-Universität zu Berlin (2007) submitted to a journal.
14. Massuthe, P., Wolf, K.: An Algorithm for Matching Non-deterministic Services with Operating Guidelines. International Journal of Business Process Integration and Management (IJBPIM) **2**(2) (2007) 81–90
15. Schmidt, K.: Controllability of Open Workflow Nets. In Desel, J., Frank, U., eds.: Enterprise Modelling and Information Systems Architectures. Volume P-75 of Lecture Notes in Informatics (LNI)., Bonn, Entwicklungsmethoden für Informationssysteme und deren Anwendung (EMISA, RWTH Aachen), Köllen Druck+Verlag GmbH (2005) 236–249
16. Lohmann, N., Massuthe, P., Wolf, K.: Behavioral Constraints for Services. In Alonso, G., Dadam, P., Rosemann, M., eds.: Business Process Management, 5th International Conference, BPM 2007, Brisbane, Australia, September 24-28, 2007, Proceedings. Volume 4714 of Lecture Notes in Computer Science., Springer-Verlag (2007) 271–287
17. Vogler, W.: Modular Construction and Partial Order Semantics of Petri Nets. Volume 625 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, Heidelberg, New York (1992)
18. Aalst, W.M.P.v.d., Basten, T.: Inheritance of Workflows: An Approach to Tackling Problems Related to Change. Theoretical Computer Science **270**(1-2) (2002) 125–203
19. Basten, T., Aalst, W.M.P.v.d.: Inheritance of Behavior. Journal of Logic and Algebraic Programming **47**(2) (2001) 47–145
20. Bonchi, F., Brogi, A., Corfini, S., Gadducci, F.: A Behavioural Congruence for Web Services. In Arbab, F., Sirjani, M., eds.: International Symposium on Fundamentals of Software Engineering, International Symposium, FSEN 2007, Tehran, Iran, April 17-19, 2007, Proceedings. Volume 4767 of Lecture Notes in Computer Science., Springer-Verlag (2007) 240–256
21. Bordeaux, L., Salaün, G., Berardi, D., Mecella, M.: When are Two Web Services Compatible? In Shan, M., Dayal, U., Hsu, M., eds.: Technologies for E-Services, 5th International Workshop, TES 2004. Volume 3324 of Lecture Notes in Computer Science., Springer-Verlag (2004) 15–28
22. Beyer, D., Chakrabarti, A., Henzinger, T.: Web service interfaces. In Ellis, A., Hagino, T., eds.: Proceedings of the 14th international conference on World Wide Web, WWW 2005, ACM (2005) 148–159
23. Benatallah, B., Casati, F., Toumani, F.: Representing, Analysing and Managing Web Service Protocols. Data Knowl. Eng. **58**(3) (2006) 327–357
24. Cerná, I., Vareková, P., Zimmerová, B.: Component Substitutability via Equivalencies of Component-Interaction Automata. In: Proceedings of the International Workshop on Formal Aspects of Component Software (FACS'06), Amsterdam, The Netherlands, Elsevier ENTCS (2007) 39–55
25. Castagna, G., Gesbert, N., Padovani, L.: A Theory of Contracts for Web Services. SIGPLAN Not. **43**(1) (2008) 261–272

26. Bravetti, M., Zavattaro, G.: Contract Based Multi-party Service Composition. In Arbab, F., Sirjani, M., eds.: International Symposium on Fundamentals of Software Engineering, International Symposium, FSEN 2007, Tehran, Iran, April 17-19, 2007, Proceedings. Volume 4767 of Lecture Notes in Computer Science., Springer-Verlag (2007) 207–222
27. Fournet, C., Hoare, C.A.R., Rajamani, S.K., Rehof, J.: Stuck-Free Conformance. In Alur, R., Peled, D., eds.: Computer Aided Verification, 16th International Conference, CAV 2004, Boston, MA, USA, July 13-17, 2004, Proceedings. Volume 3114 of Lecture Notes in Computer Science., Springer-Verlag (2004) 242–254
28. Andrews, T., Qadeer, S., Rajamani, S.K., Rehof, J., Xie, Y.: Zing: A Model Checker for Concurrent Software. In: Computer Aided Verification, 16th International Conference, CAV 2004, Boston, MA, USA, July 13-17, 2004, Proceedings. Volume 3114 of Lecture Notes in Computer Science., Springer-Verlag (2004) 484–487
29. Sharygina, N., Chaki, S., Clarke, E., Sinha, N.: Dynamic Component Substitutability Analysis. In Fitzgerald, J., Hayes, I., Tarlecki, A., eds.: FM 2005: Formal Methods, International Symposium of Formal Methods Europe, Proceedings. Volume 3582 of Lecture Notes in Computer Science., Springer-Verlag (2005) 512–528
30. Pathak, J., Basu, S., Honavar, V.: On Context-Specific Substitutability of Web Services. In: 2007 IEEE International Conference on Web Services (ICWS 2007), July 9-13, 2007, Salt Lake City, Utah, USA, IEEE Computer Society (2007) 192–199